Directed Evolution in Live Coding Music Performance

Sandeep Dasari and Jason Freeman

Georgia Institute of Technology
sdasari38@gatech.edu

Abstract. Genetic algorithms are extensively used to understand, simulate, and create works of art and music. In this paper, a similar approach is taken to apply basic evolutionary algorithms to perform music live using code. Often considered an improvisational or experimental performance, live coding music comes with its own set of challenges. Genetic algorithms offer potential to address these long-standing challenges. Traditional evolutionary applications in music focused on novelty search to create new sounds, sequences of notes or chords, and effects. In contrast, this paper focuses on live performance to create **directed** evolving musical pieces. The paper also details some key design decisions, implementation, and usage of a novel genetic algorithm API created for a popular live coding language.

Keywords: evolution, genetic algorithms, live coding music, AI

1 Introduction

In the past few years, there has been a significant amount of research in evolution, genetic algorithms, and artificial intelligence due to developments in computing power and accessibility. Using these powerful tools to understand, simulate, and create art is an exciting and meaningful endeavor explored by inter-disciplinary researchers in visual art and music (Johnson & Romero, 2002). While genetic algorithms are applied in novelty search frequently, this paper presents an approach to apply evolutionary algorithms to generate and control *musical evolutions* in a live performance.

Live coding music is a unique tool of musical expression enabling an artist to interact with their instrument through code. The practice enables the artist to efficiently control a vast list of parameters through algorithms as opposed to traditional music software with MIDI controllers, keyboards, and other interfaces. Due to extreme granularity of control and slow interaction through code, live coding music poses a few complex yet interesting challenges (Collins, McLean, Rohrhuber, & Ward, 2003). The community has seen some key contributions (McLean, Griffiths, Collins, & Wiggins, 2010; Lee & Essl, 2014) towards solving these problems and consequently discovered novel approaches to music composition and performance (Kirkbride, 2017). In this direction, this paper proposes an application of genetic algorithms to solve two specific challenges: long setup

times to reach a meaningful state of music, and complex interaction to create evolving pieces of music.

This paper introduces **evomusic**, a genetic algorithm system for live coding music. With evomusic, an experienced live coder will be able to specify a direction of musical evolution through a list of parameters and build layers of evolving musical pieces. To achieve a level of control and predictability in live performance, the problem domain is tweaked to determine a series of evolutions in a path from a source node to a destination node. evomusic enables intermediate and advanced live coders to access and control genetic algorithms during a performance. This can be described as an API to generate a series of directed evolutions that augment a live coder's setup. Lastly, the project is an open-source contribution to an existing live coding language FoxDot (Kirkbride, 2016) and a popular open-source hackable text editor Atom, encouraging the use and development of this system.

2 Related work

Genetic programming (GP) is a technique of evolving programs using genetic algorithms. Application of GP in creative arts and music has upsurged from a small inter-disciplinary experiment to a complete area of research interest including major conferences like EvoMUSART. Loughran and O'Neill (2020) presents a detailed review of a large list of applications of GP in creating music. The article describes a common motivation for applying these algorithms in music: to appreciate the slow sonic evolution of a piece. The authors explore the practice and discuss the inevitable unpredictability involved in the generation process.

2.1 Applications in Music

Applications of evolution in music have largely been artistic or experimental (exploring possibilities of creating or evolving new art). Examples of this work can be seen in research in computational creativity (Boden, 2009) and creative composition and computer interaction (Gartland-Jones, 2003). In contrast, a few performance-oriented tools have been used in live performance to improvise with jazz musicians (J. Biles, 1994) and autonomous evolution of piano performances (Dahlstedt, 2007). Thus, the practice of generating music with GP has been standardized in the process yet diversified in application. The practice of live coding itself has seen applications of generative algorithms in performance including some detailed experiences documented by accomplished live coding duo *aa-cell* in Collins et al. (2003). Drawing from these experiences and research, code creation using code generators and a balance between abstraction and complete control of the generative process are two main directing design principles used in the design and development of evomusic.

Pursuing an exploratory endeavor, Hickinbotham and Stepney (2016) presents a networked environment with multiple live instances of genetically generated

code. The system is an exciting experiment in tapping into the powerful Haskell based live coding language TidalCycles (McLean & Wiggins, 2010) and a networked environment ExtraMuros (Ogborn, Tsabary, Jarvis, Cárdenas, & McLean, 2015) wrapped with live coding interactions in a clean and user-friendly interface. Evaluating the fitness of a mutation depends on the execution of the mutation to generate audio and can be a slow system to create expressive mutations. The author acknowledges this issue: "A key problem that needs to be addressed is the relatively low fitness of newly-generated patterns compared with patterns that exist in the population". The system is a great tool for novelty searches and does not set out to solve our identified challenges of slow setup time and control. In contrast, evomusic proposes a directed evolution approach in a single text editor with detailed access and control over the list of genetic evolution parameters.

2.2 Representation

In order to generate evolutions of a section of music, the algorithm needs to be fed an understandable representation of the section. The networked system, Extramuros uses a recursive parser to understand a subset of the Tidal language and generates new code by using crossover operations on the branches of the parse tree (mutations are completely avoided in this system). A similar parser-generator approach is used in this paper to deal with understanding and representing the state of a musical piece.

2.3 Why Directed Evolution

While advanced fitness functions and heuristics are ideal for creating refined evolutions that follow music theory, the application of these algorithms in live coding environments is blurred and often difficult to understand. The implementation of evomusic takes a much simpler approach that can be visualized by the live coder. Directed evolution is a naive concept of moving or in GP terms *evolving* from a source node towards a target node. Although not traditional Darwinian evolution, the approach automates evolutions towards a goal solving the famous *fitness bottleneck* described in J. A. Biles (2001). This simple yet efficient feature of evomusic makes it ideal to create evolving musical layers which is often reflected in music production as automation lanes and sections of music. Evolving a piece of music in this way is not entirely novel. Horner and Goldberg (1991) explores music composition through a paradigm called thematic bridging. In their own words, the authors define **thematic bridging** as "the transformation of an initial musical pattern to some final pattern over a specific duration of time". The results of this work were interesting but largely limited to sequences of notes. Access to a wide range of musical parameters is simplified through live coding languages, making directed evolution a perfect tool for composing simple sequences of code with a vast range of sonic possibilities.

Research in deep learning with digital audio to generate and explore computer music in NSynth (Engel et al., 2017) and MuseNet (Dong, Hsiao, Yang, & Yang, 2018) produced great results compared to a regular genetic algorithm,

however, using deep learning in live performance is unsuitable. Firstly, the computation power required to generate pieces on the fly while performing is inadequate. Secondly, the lack of active control in the generation i.e. deciphering the deep learning parameters, the complex mapping of the hidden layers of a neural network makes it impractical for live performance. In this regard, a genetic algorithm is simpler to understand (up to a certain degree of abstraction) and control, making evomusic an instrument to control generative music rather than an autonomous system.

3 Design Narrative

A considerable effort was spent in understanding the requirements of the system, intended audience, design, and development of the software. One of the key contributions of the system is to deliver a basic yet efficient and usable tool to control meaningful evolutions in live coding. Studying documented experiences and approaches of live coding practitioners helped in understanding two basic challenges that were taken into consideration during the design phase.

3.1 Challenges in Live Coding Music

Setup time: Often live coding performers starting from scratch are forced to have a few seconds of silence or a small piece of repetitive music playing in the background as they set up the initial code for their performance. While evomusic does not propose to eliminate the initial setup time, it enables the artist to create instant evolving layers of music so that the artist can then focus on setting up musical elements in the foreground. In Brown and Sorensen (2009), the authors describe the need and usage of generative algorithms. In their own words: "While it is possible to trigger sound events directly while live coding, it is much more efficient to create generative processes that autonomously make music, freeing the performer to build or modify code for the next stage of the performance".

Control: Having access to a wide palette of parameters to readily tweak is a strong feature of live coding, but the fine-grained control for each parameter can be overwhelming. Brown and Sorensen (2009) suggests the balancing of control and surprise is a constant challenge for generative sound artists and is better handled by the performer than a computational agent. Although the authors are cautious about autonomous generative agents, the requirement of human control guiding generative algorithms is well established and used as a design guideline for evomusic.

3.2 Software

The system uses some pre-existing standard tools in live coding music: text editor, Atom[1]; Python Library for live coding, FoxDot; and an audio server, Su-

[1] an open-source hackable text editor: https://atom.io

percollider[2]. The functionalities of evomusic are built into an independent Atom package. For a live coder, an interactive visual text editor is critical during a performance. To start a new evolution, a Python method *evolve* is executed using a key-binding *Cmd/Ctrl+E*. This is set apart from FoxDot's inherent code execution key-binding *Cmd/Ctrl+return* to clearly differentiate between executing a repetitive piece of code and starting an evolution of code. The parameters of this method are parsed and used to set up the system for evolutions. In FoxDot (Kirkbride, 2016), any sound is played through a *Player*(a SynthDef in SuperCollider). A Player object is a sequencer that accepts a wide range of parameters to generate repetitions of music. In order to start an evolution, a source player and a destination player object are mandatory parameters to the evolve method. Once an evolution is started, a new function is created, updated, and executed at periodic intervals.

4 Implementation

4.1 Forming source and destination genomes

The parser evaluates each Player object into a parse tree that represents the functionality of the Player Object and will be referred to as a genome. The result of the parser is a JSON representation of the Player object displayed in the Parser phase of Fig. 1. This representation is fed as input to the Genetic Algorithm API. Traditional genetic algorithms convert all the parameters and values into a single list of numbers. This leads to unpredictable results unsuitable for live performance. For example, Player *p1*

```
p1 >> pads(degree=[4,6,7,2], dur=[2,2,4,0.5], amp=0.2, lpf=5000)
```

generates a genome [**4,6,7,2,2,2,4,0.5**]. Evolving this genome may create multiple generations of the following kind: [**4.2,6,8.4...**] where the degree(pitch) is a microtonal value. In evomusic, this behavior can be avoided by setting stepSize (quantization value for the current evolution) accordingly. Lastly, it must be acknowledged that the parser is still far from an exhaustive parsing of the FoxDot syntax. FoxDot pattern objects like *PDur*, *PRand*, *PStep* etc. are not currently supported by the parser.

4.2 Implementing genetic algorithms

Genetic algorithms in JavaScript were implemented using an NPM package: geneticalalgorithm[3]. The native mutation, crossover and fitness functions were overridden to support the currently presented solution. The generated source genome from the previous phase is used to populate the first generation of a genetic algorithm. Default parameters for the evolve function were determined

[2] platform for audio synthesis: https://supercollider.github.io
[3] `https://www.npmjs.com/package/geneticalgorithm`

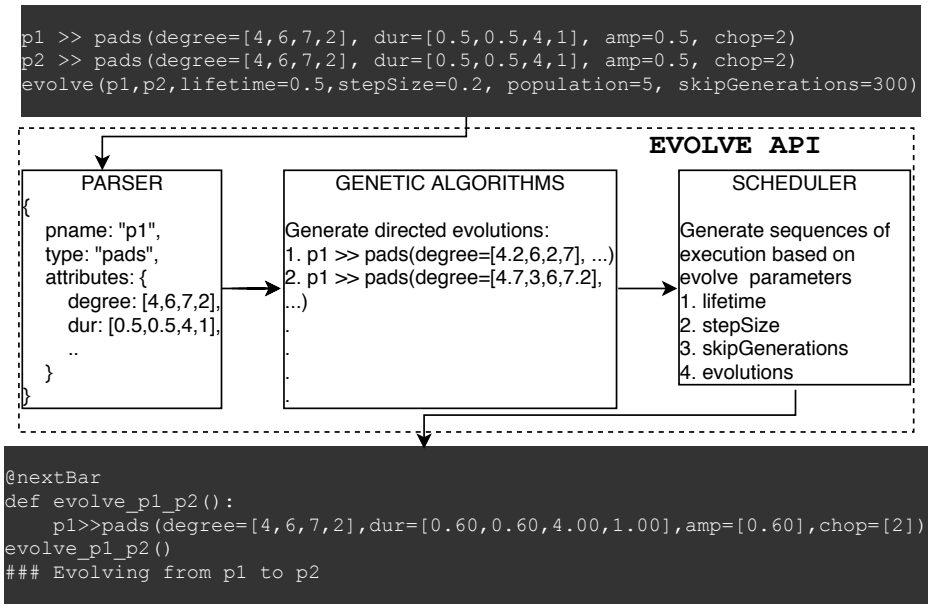

Fig. 1. Code generation process between text editor and evolve API displayed as a flow diagram

through experimentation.

Mutation and Crossover: Mutation multiplies the constant *mutationAmount* to create new mutations of code. Crossover applies only when the input genome is a list of multiple values. The values of the list are swapped based on the *crossoverAmount*.
Fitness Function: To facilitate a directed evolution from a source phenotype to a destination phenotype, a trivial distance metric function, Euclidean distance, is used.

4.3 Implementing text editor interactions

In order to control evolutions, three main operations are supported:

1. Selection and Generation of a block of code: Upon using the evolve keymap **Cmd/Ctrl+E**, a new Python method is generated starting from the source genome and executed asynchronously. *@nextBar* is a FoxDot specific decorator that ensures that the generated function is synchronised to execute at the beginning of the next musical bar.
2. Update player attributes: Player attributes can be overridden if needed to allow the coder expressivity beyond the generated code.
3. Stop an evolution: An essential functionality for any musical system is to be able to start and stop the system at will. Any evolution can be stopped using the key map **Cmd/Ctrl+Z**.

```
def evolve_c3_c4():
    c3>>space(degree=[3,3,0,3],amp=[0.08],tremolo=[1])
    c3.tremolo = 0;           // override generated attributes
evolve_c3_c4()
#|########## Evolving from c3 to c4
```

Fig. 2. Generated function evolving Player c3 to c4

Visual feedback is added to display the progression by appending #'s in the feedback string to reflect the states being executed. Additionally, the generated function and the current evolution # is flashed in blue for 300ms to help the coder keep time and control other code actions.

5 Results and Challenges

Observations drawn from personal experiences are presented to illustrate the usage and capability of the system. The experiments or performances, in this case, were recorded as live coding sessions and can be accessed at the project repository[4]. The experiments were focused largely on expression, control, and understanding the limitations of the system.

5.1 Expression and Control

Expression is evaluated based on the classic ceiling-floor discussion (Jack, Harrison, Morreale, & McPherson, 2018) in evaluating music instruments. The system enables quick real-time interactions to start, stop and customize the generation irrespective of the sound sources (limited only by the coding language and the coder). The expressive possibilities of this system are rich and endless, giving it a high ceiling of expression. However, it should be acknowledged that this system is intended for intermediate and advanced live coders. The user is expected to set up the system, understand and have experimented with FoxDot syntaxes, and have knowledge of genetic algorithms and the parameters, making it a high-floor instrument to master and perform.

During the performances, a key observation made was that the usage of this system requires a clear structure of the piece to evolve in advance, as you are expected to specify the end destination of the evolution. This need for a structure can be a tool to write expressive improvisations for artists or search for motivation between two musical nodes narrowed down by the artist. The ability to run multiple parallel threads of evolutions is key to performing live music. This enables the live coder to create complex textures and layers in the track.

5.2 Limitations

Text editor interactions are limited in their expressive feedback ability. It can be daunting to keep track of more than five evolutions happening within a code

[4] https://github.com/sandcobainer/evomusic/#readme

window. A common issue observed during the performance was bugs related to overwritten text buffers. Although limited to extreme situations, limitation of computing resources should also be acknowledged. Generating a long piece of more than 10000 evolutions can cause the system to crash or slow down affecting the live performance drastically.

FoxDot allows extreme ranges of musical parameters to be executed without validation. evomusic tries to solve this issue with a pre-compiled list to avoid extreme parameter ranges subjectively. Although this is required to avoid unpredictable evolutions, the beam of search for the genetic algorithms is narrowed. This can be perceived as a by-product rather than a limitation of the system. The control of the system is limited beyond a certain threshold of musical expression. For example, the user currently can start evolutions and stop them. However, the coder cannot pause an evolution or loop a section of the evolution along the way. The coder is forced to accept the unpredictability of the system, reducing the intended expressive capability.

6 Future Work

Drawing from the observations from Section 3 and Section 5, future work can be done in a few different directions. This paper lacks a structured evaluation of the system through multiple performances and users in various fields and comparison to other similar generative systems (Hickinbotham & Stepney, 2016). The API will be extended to support the traditional novelty search algorithm. The two approaches will be contrasted and discussed quantitatively and qualitatively. The system can be significantly enhanced visually and addressing the limitation of control in pausing, looping evolutions. Evaluations should also consider the usage of collaborative live coding frameworks as an interesting endeavor in evolving music. These studies are critical to understanding the true potential of the system beyond the solo performer approach.

7 Conclusion

evomusic is an initial effort to apply genetic programming in live coding music. The system enables users to perform evolutions live and is aimed to solve two main challenges in live coding: setup time and expressive control of genetic algorithms. In this regard, the system is a successful endeavor in creating an efficient framework to explore a vast array of sonic possibilities during a live performance. The limitations of the systems are key by-products that help us further understand live coding music requirements. The idea of applying continuous evolutions in real-time in live coding is an interesting research step towards musical creativity and formal live performance and is well worth exploring.

References

Biles, J. (1994, September). Genjam: A genetic algorithm for generating jazz solos. In *International Computer Music Conference (ICMC)* (Vol. 94, pp. 131–137).

Biles, J. A. (2001). Autonomous GenJam: eliminating the fitness bottleneck by eliminating fitness. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference Workshop Program, San Francisco.*

Boden, M. A. (2009). Computer models of creativity. *AI Magazine*, *30*(3), 23–23.

Brown, A. R., & Sorensen, A. (2009). Interacting with generative music through live coding. *Contemporary Music Review*, *28*(1), 17–29.

Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. *Organised sound*, *8*(3), 321–330.

Dahlstedt, P. (2007, September). Autonomous evolution of complete piano pieces and performances. In *Proceedings of Music AL Workshop.*

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C., & Yang, Y.-H. (2018). Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *AAAI Conference on Artificial Intelligence.* Retrieved from `https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17286`

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D., & Simonyan, K. (2017, July). Neural audio synthesis of musical notes with WaveNet autoencoders. In D. Precup & Y. W. Teh (Eds.), (Vol. 70, pp. 1068–1077). PMLR.

Gartland-Jones, A. (2003). MusicBlox: A Real-Time Algorithmic Composition System Incorporating a Distributed Interactive Genetic Algorithm. In S. Cagnoni et al. (Eds.), *Applications of Evolutionary Computing* (pp. 490–501). Springer Berlin Heidelberg.

Hickinbotham, S., & Stepney, S. (2016). Augmenting Live Coding with Evolved Patterns. In C. Johnson, V. Ciesielski, J. Correia, & P. Machado (Eds.), *Evolutionary and Biologically Inspired Music, Sound, Art and Design* (pp. 31–46). Cham: Springer International Publishing.

Horner, A., & Goldberg, D. (1991). Genetic algorithms and computer-assisted music composition. In *International Computer Music Conference (ICMC).*

Jack, R. H., Harrison, J., Morreale, F., & McPherson, A. P. (2018). Democratising DMIs: the relationship of expertise and control intimacy. In *NIME* (pp. 184–189).

Johnson, C., & Romero, J. (2002). Genetic Algorithms in Visual Art and Music. *Leonardo*, *35*, 175-184.

Kirkbride, R. (2016, October). Foxdot: Live coding with python and supercollider. In *Proceedings of the International Conference on Live Interfaces* (pp. 194–198).

Kirkbride, R. (2017). Troop: a collaborative tool for live coding. In *Proceedings of the 14th Sound and Music Computing Conference* (pp. 104–9).

Lee, S. W., & Essl, G. (2014). Communication, control, and state sharing in networked collaborative live coding. *Ann Arbor*, *1001*, 48109–2121.

Loughran, R., & O'Neill, M. (2020). Evolutionary music: applying evolutionary computation to the art of creating music. *Genetic Programming and Evolvable Machines*, *21*, 55–85.

McLean, A., Griffiths, D., Collins, N., & Wiggins, G. (2010). Visualisation of live code. *Electronic Visualisation and the Arts (EVA 2010)*, 26–30.

McLean, A., & Wiggins, G. (2010). Tidal–pattern language for the live coding of music. In *Proceedings of the 7th sound and music computing conference.*

Ogborn, D., Tsabary, E., Jarvis, I., Cárdenas, A., & McLean, A. (2015). Extramuros: making music in a browser-based, language-neutral collaborative live coding environment. In *Proceedings of the First International Conference on Live Coding* (pp. 163–69).